Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и

системное программирование

ОТЧЁТ

к лабораторной работе №6

на тему

Средства синхронизации и взаимного исключения (Windows). Изучение и использование средств синхронизации и взаимного исключения

Выполнил: студент группы 153503

Киселёва Елизавета Андреевна

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc149265314)

[2 Краткие теоретические сведения 4](#_Toc149265315)

[2.1 Мьютексы в *Windows* 4](#_Toc149265316)

[2.2 Семафоры в *Windows* 4](#_Toc149265317)

[2.3 Другие средства синхронизации и взаимного исключения 5](#_Toc149265318)

[3 Результаты выполнения лабораторной работы 7](#_Toc149265319)

[Выводы 8](#_Toc149265320)

[Список использованных источников 9](#_Toc149265321)

[Приложение А (обязательное) Листинг кода 10](#_Toc149265322)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание приложения, использующего средства синхронизации (мьютексы, семафоры) для координирования работы потоков. Для этого будет создан текстовый редактор, в котором сохранение и загрузка файла происходят в разных потоках с помощью мьютексов.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 2.1 Мьютексы в *Windows*

Мьютексы – это объекты синхронизации, которые предоставляют механизм взаимного исключения для потоков. Только один поток может владеть мьютексом в определенный момент времени. Мьютексы предоставляют механизм контроля доступа к общим ресурсам, таким как разделяемые данные или критические секции кода, и являются важной частью средств синхронизации в многозадачных приложениях.

Основной целью мьютексов является предотвращение состязательного доступа, когда несколько потоков одновременно пытаются изменять общие данные. Для этого мьютексы обеспечивают взаимное исключение, гарантируя, что только один поток может владеть мьютексом в конкретный момент времени.

В *Windows* существует два основных типа мьютексов: двоичные (*binary*) и счетные (*counting*). Двоичный мьютекс имеет два состояния – захвачен и освобожден. Счетный мьютекс позволяет считать количество успешных захватов и освобождений, что может быть полезно в определенных сценариях.

Для работы с мьютексами в *Win32 API* используются функции, такие как *CreateMutex* для создания мьютекса, *WaitForSingleObject* для захвата, *ReleaseMutex* для освобождения, и *CloseHandle* для закрытия дескриптора мьютекса после завершения работы.

Использование мьютексов важно для обеспечения безопасного и согласованного доступа к данным в многозадачных приложениях, где несколько потоков могут конкурировать за общие ресурсы. [1]

## 2.2 Семафоры в *Windows*

Семафоры в операционной системе *Windows* представляют собой объекты синхронизации, используемые для управления доступом к общим ресурсам в многозадачных приложениях. Они предоставляют эффективный механизм счетчика, который контролирует количество потоков, имеющих доступ к определенному ресурсу.

Основное предназначение семафоров заключается в том, чтобы регулировать доступ к определенному количеству ресурсов или критических секций кода. В отличие от мьютексов, семафоры могут поддерживать одновременный доступ нескольких потоков, в зависимости от установленного счетчика.

В *Windows* можно выделить два основных типа семафоров: двоичные (*binary*) и счетные (*counting*). Двоичный семафор может принимать только два значения — 0 или 1, и обычно используется для регулирования доступа к единственному ресурсу. Счетный семафор содержит счетчик, который увеличивается и уменьшается в процессе захвата и освобождения ресурсов.

Для работы с семафорами в *Win32 API* используются функции, такие как *CreateSemaphore* для создания семафора, *WaitForSingleObject* или *WaitForMultipleObjects* для захвата, *ReleaseSemaphore* для освобождения, и *CloseHandle* для закрытия дескриптора семафора после завершения работы.

Использование семафоров является эффективным способом управления доступом к общим ресурсам в параллельных приложениях, где потоки конкурируют за доступ к определенному количеству ресурсов. [2]

## 2.3 Другие средства синхронизации и взаимного исключения

События в операционной системе *Windows* представляют собой объекты синхронизации, используемые для уведомления одного или нескольких потоков о возникновении определенного события. Эти объекты являются ключевым механизмом для синхронизации потоков и позволяют им взаимодействовать в сценариях, где один поток ожидает сигнала от другого о завершении определенного действия.

События обладают двумя состояниями: сигнальным и несигнальным. Потоки могут ожидать события, блокируясь до момента его сигнала. Когда событие сигнализируется, один или несколько потоков продолжают выполнение.

Критическая секция представляет собой участок кода, который требует взаимного исключения для обеспечения безопасного доступа к общим данным. Это используется для предотвращения состязательного доступа нескольких потоков к общим ресурсам, что может привести к непредсказуемым результатам или ошибкам.

Для взаимодействия между процессами в *Windows* используются объекты синхронизации, такие как файл-маппинг, события и семафоры. Эти средства позволяют процессам обмениваться данными и синхронизировать свою работу.

*Reader-Writer Locks* предоставляют механизм взаимного исключения, который разрешает нескольким потокам читать данные одновременно, но блокирует запись во время чтения. Это может улучшить производительность в тех случаях, когда чтение является более распространенной операцией, чем запись.

Атомарные операции гарантируют, что операции над общими данными выполняются целиком, без вмешательства других потоков. Это важно для предотвращения состязательных ситуаций, когда несколько потоков могут пытаться изменить общие данные одновременно.

Эти механизмы синхронизации являются важной частью разработки параллельных и многозадачных приложений в операционной системе *Windows*, обеспечивая правильное взаимодействие между потоками и процессами. [3]

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы был разработан текстовый редактор, в котором сохранение в файл происходит в одном потоке, а загрузка файла – в другом с использованием мьютексов. (Рисунок 3.1).

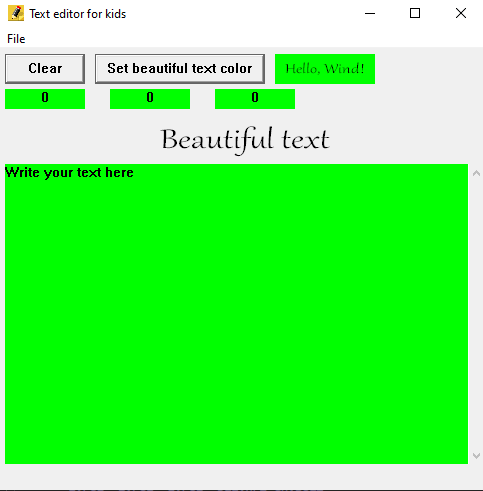
****

Рисунок 3.1 – Результат работы программы

В меню *File* можно сохранить файл, загрузить сохраненный файл, очистить поле ввода и выйти из приложения (Рисунок 3.2).

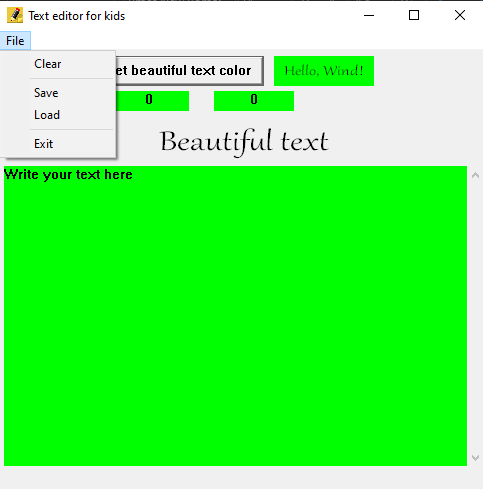


Рисунок 3.2 – Результат работы программы при установленном английском языке клавиатуры

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы был разработан текстовый редактор, в котором сохранение в файл происходит в одном потоке, а загрузка файла в другом с использованием мьютексов. Проект включает в себя создание графического окна приложения, обработку оконных сообщений, работу с потоками, использование мьютексов, работу с файлами. Результатом стал рабочий текстовый редактор.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Microsoft Learn. Использование объектов мьютекса [Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/sync/using-mutex-objects.

[2] Microsoft Learn. Использование объектов семафора[Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/sync/using-semaphore-objects.

[3] Microsoft Learn [Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/sync/using-critical-section-objects.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**Source.cpp**

#include <Windows.h>

#include <string>

#include "Header.h"

#include "resource.h"

#include <thread>

// Глобальный флаг для отслеживания сохранения файла

bool fileSaved = false;

HANDLE mutex; // Глобальный мьютекс для синхронизации доступа к файлу

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshow) {

// Инициализация мьютекса

mutex = CreateMutex(NULL, FALSE, NULL);

if (mutex == NULL) {

// Обработка ошибки при создании мьютекса

MessageBox(NULL, L"Failed to create mutex.", L"Error", MB\_ICONERROR | MB\_OK);

return -1;

}

fontRectangle = CreateFontA(

60, 20, 0, 0, FW\_MEDIUM,

FALSE, FALSE, FALSE, DEFAULT\_CHARSET,

OUT\_OUTLINE\_PRECIS, CLIP\_DEFAULT\_PRECIS, ANTIALIASED\_QUALITY,

FF\_DECORATIVE, "SpecialFont"

);

fontStatic = CreateFontA(

30, 10, 0, 0, FW\_MEDIUM,

FALSE, FALSE, FALSE, DEFAULT\_CHARSET,

OUT\_OUTLINE\_PRECIS, CLIP\_DEFAULT\_PRECIS, ANTIALIASED\_QUALITY,

FF\_DECORATIVE, "SpecialStaticFont"

);

// Создание класса окна WNDCLASS и задание его параметров

WNDCLASS SoftwareMainClass = NewWindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_HAND), hInst,

LoadIcon(hInst, MAKEINTRESOURCE(IDI\_ICON1)), L"MainWndClass", SoftwareMainProcedure);

// Регистрация класса окна

if (!RegisterClassW(&SoftwareMainClass)) { return -1; }

MSG SoftwareMainMessage = { 0 };

// параметр WS\_OVERLAPPEDWINDOW добавляет кнопки размер, свернуть, закрыть на окно + перемещение окна + стандартные бордюр и рамка + системное меню

CreateWindow(L"MainWndClass", L"Text editor for kids", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 500, 500, NULL, NULL, NULL, NULL);

while (GetMessage(&SoftwareMainMessage, NULL, NULL, NULL))

{

TranslateMessage(&SoftwareMainMessage);

DispatchMessage(&SoftwareMainMessage);

}

// Закрываем мьютекс при завершении программы

CloseHandle(mutex);

return 0;

}

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure) {

WNDCLASS NWC = { 0 };

NWC.hCursor = Cursor;

NWC.hIcon = Icon;

NWC.hInstance = hInst;

NWC.lpszClassName = Name;

NWC.hbrBackground = BGColor;

NWC.lpfnWndProc = Procedure;

return NWC;

}

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg)

{

case WM\_COMMAND:

switch (wp) {

case OnClearField:

SetWindowTextA(hEditControl, "");

break;

case OnReadColor:

colorR = GetDlgItemInt(hWnd, DigIndexColorR, FALSE, false);

colorG = GetDlgItemInt(hWnd, DigIndexColorG, FALSE, false);

colorB = GetDlgItemInt(hWnd, DigIndexColorB, FALSE, false);

//brushRectangle = CreateSolidBrush(RGB(colorR, colorG, colorB));

fontColor = RGB(colorR, colorG, colorB);

RedrawWindow(hWnd, NULL, NULL, RDW\_UPDATENOW | RDW\_INVALIDATE);

break;

case OnSaveFile:

if (GetSaveFileNameA(&ofn)) { SaveDataThread(filename); }

break;

case OnLoadFile:

if (GetOpenFileNameA(&ofn)) { LoadDataThread(filename); }

break;

case OnExitSoftware:

PostQuitMessage(0);

break;

default:

break;

}

break;

case WM\_PAINT:

BeginPaint(hWnd, &ps);

SetBkMode(ps.hdc, TRANSPARENT);

SetTextColor(ps.hdc, fontColor);

SelectObject(ps.hdc, fontRectangle);

// DT\_NOCLIP - если текст не влезет в прямоугольник, то выйдет за его пределы и будет виден

DrawTextA(ps.hdc, "Beautiful text", 15, &windowRectangle, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER | DT\_NOCLIP);

EndPaint(hWnd, &ps);

break;

case WM\_CREATE:

MainWndAddMenus(hWnd);

MainWndAddWidgets(hWnd);

SetOpenFileParams(hWnd);

SendMessageA(hStaticControl, WM\_SETFONT, (WPARAM)fontStatic, TRUE);

break;

case WM\_CTLCOLORSTATIC:

hdcStatic = (HDC)wp;

SetBkColor(hdcStatic, RGB(0, 255, 0));

return (INT\_PTR)CreateSolidBrush(RGB(0, 255, 0));

break;

case WM\_CTLCOLOREDIT:

hdcStatic = (HDC)wp;

SetBkColor(hdcStatic, RGB(0, 255, 0));

return (INT\_PTR)CreateSolidBrush(RGB(0, 255, 0));

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default: return DefWindowProc(hWnd, msg, wp, lp);

}

}

void MainWndAddMenus(HWND hWnd) {

HMENU RootMenu = CreateMenu();

HMENU SubMenu = CreateMenu();

HMENU SubActionMenu = CreateMenu();

AppendMenu(SubMenu, MF\_STRING, OnClearField, L"Clear");

AppendMenu(SubMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenu(SubMenu, MF\_STRING, OnSaveFile, L"Save");

AppendMenu(SubMenu, MF\_STRING, OnLoadFile, L"Load");

AppendMenu(SubMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenu(SubMenu, MF\_STRING, OnExitSoftware, L"Exit");

AppendMenu(RootMenu, MF\_POPUP, (UINT\_PTR)SubMenu, L"File");

SetMenu(hWnd, RootMenu);

}

void MainWndAddWidgets(HWND hWnd) {

// WS\_VISIBLE - элемент видим по дефолту, WS\_CHILD - элемент является дочерним от базового окна(наодится внутри этого окна)

hStaticControl = CreateWindowA("static", "Hello, Wind!", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 275, 5, 100, 30, hWnd, NULL, NULL, NULL);

windowRectangle = { 5 + 480, 70, 5, 110 };

hEditControl = CreateWindowA("edit", "Write your text here", WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | WS\_VSCROLL, 5, 115, 480, 300, hWnd, NULL, NULL, NULL);

CreateWindowA("edit", "0", WS\_VISIBLE | WS\_CHILD | ES\_CENTER | ES\_NUMBER, 5, 40, 80, 20, hWnd, (HMENU)DigIndexColorR, NULL, NULL);

CreateWindowA("edit", "0", WS\_VISIBLE | WS\_CHILD | ES\_CENTER | ES\_NUMBER, 110, 40, 80, 20, hWnd, (HMENU)DigIndexColorG, NULL, NULL);

CreateWindowA("edit", "0", WS\_VISIBLE | WS\_CHILD | ES\_CENTER | ES\_NUMBER, 215, 40, 80, 20, hWnd, (HMENU)DigIndexColorB, NULL, NULL);

CreateWindowA("button", "Clear", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 5, 5, 80, 30, hWnd, (HMENU)OnClearField, NULL, NULL);

CreateWindowA("button", "Set beautiful text color", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 95, 5, 170, 30, hWnd, (HMENU)OnReadColor, NULL, NULL);

}

void SaveData(LPCSTR path) {

// Захватываем мьютекс перед доступом к файлу

WaitForSingleObject(mutex, INFINITE);

HANDLE FileToSave = CreateFileA(

path,

GENERIC\_WRITE,

0,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL

);

if (FileToSave != INVALID\_HANDLE\_VALUE) {

int saveLength = GetWindowTextLengthA(hEditControl) + 1;

char\* data = new char[saveLength];

saveLength = GetWindowTextA(hEditControl, data, saveLength);

DWORD bytesIterated;

WriteFile(FileToSave, data, saveLength, &bytesIterated, NULL);

CloseHandle(FileToSave);

fileSaved = true;

}

// Освобождаем мьютекс после завершения операции

ReleaseMutex(mutex);

}

void LoadData(LPCSTR path) {

// Захватываем мьютекс перед доступом к файлу

WaitForSingleObject(mutex, INFINITE);

HANDLE FileToLoad = CreateFileA(

path,

GENERIC\_READ,

0,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL

);

if (FileToLoad != INVALID\_HANDLE\_VALUE) {

DWORD bytesIterated;

ReadFile(FileToLoad, Buffer, TextBufferSize, &bytesIterated, NULL);

// Очищаем поле ввода, если файл был сохранен

if (fileSaved) {

SetWindowTextA(hEditControl, "");

fileSaved = false;

}

SetWindowTextA(hEditControl, "");

// Записываем данные из файла в поле ввода

SetWindowTextA(hEditControl, Buffer);

CloseHandle(FileToLoad);

}

// Освобождаем мьютекс после завершения операции

ReleaseMutex(mutex);

}

void SetOpenFileParams(HWND hWnd) {

// Обнуляем структуру ofn (OPENFILENAME) перед использованием

ZeroMemory(&ofn, sizeof(ofn));

// Устанавливаем размер структуры OPENFILENAME

ofn.lStructSize = sizeof(ofn);

// Устанавливаем окно-владелец для диалогового окна выбора файла

ofn.hwndOwner = hWnd;

// Устанавливаем указатель на строку, в которой будет сохранено имя выбранного файла

ofn.lpstrFile = filename;

// Устанавливаем максимальную длину имени файла, которую может принять указатель lpstrFile

ofn.nMaxFile = sizeof(filename);

// Устанавливаем фильтр файлов для диалогового окна (только .txt файлы)

ofn.lpstrFilter = "\*.txt";

// Устанавливаем указатель на строку, в которой будет сохранено имя выбранного файла без расширения

ofn.lpstrFileTitle = NULL;

// Устанавливаем максимальную длину имени файла без расширения

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = "C:/Users/Acer/Desktop/films/osisp-assets/files";

// Устанавливаем флаги для диалогового окна выбора файла (директория должна существовать и файл должен существовать)

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

}

void SaveDataThread(LPCSTR path) {

std::thread saveThread(SaveData, path);

saveThread.detach(); // Отсоединяем поток, чтобы он работал асинхронно

}

void LoadDataThread(LPCSTR path) {

std::thread loadThread(LoadData, path);

loadThread.detach(); // Отсоединяем поток, чтобы он работал асинхронно

}